CMSI 371-01
Computer Graphics
Spring 2016

Assignment 0329b Feedback

All caps are released with the outcomes in this assignment because a sufficient amount of functionality will have been reached here.

Chris Franco

cf7 / spe861@gmail.com

*Notes while running (high-priority notes are marked with ***):*

- Scene shows definite signs of transforms in action, with multiple uses.
- But with all that power…why still the tiny square viewport?
- Matrix unit test suite runs but has camera matrix failures. So that needs to be cleaned up, whether it's the test that needs to be changed or the code that needs to be fixed (due to a bug).

Code review (refer to http://lmucs.github.io/hacking-guidelines/ for code-review abbreviations):

1. The `Matrix` library certainly covers all of the bases, though the overall design has some flaws. First, the conversion to WebGL-compliant format: I can see how `glFormat` is at the heart of it, but why isn't the wrapping around a `Float32Array`? That pretty much happens all the time, yet in the code it is repeated. Also, why does this code live on the level of *hello-webgl-again.js*—this is the "app" level, and should only contain code and data pertaining specifically to the scene. Imagine yourself being asked to create a whole new scene using your libraries. If there is any code that you will end up copying over from *hello-webgl-again.js*, that is a sign that this code does not belong at the app-level but probably should live in a reusable library (whether `Shape`, or `Matrix`, or a whole new object). `glFormat` strikes me as one of those functions. (*+2a*, *+2b*, *+4a*, *4b*)
2. Similarly, the transformation operations also strike me as being unnecessarily copied if you are to create a new scene. I can see how you might not want to "mix" this with the `Shape` object, but at the very least it should probably in some bridge library that makes use of both `Shape` and `Matrix`. Again, think of what it would take to create a whole new scene file. The new code should only contain whatever is specific to the new scene. So overall, even though the functionality is there, there are aspects of how they are organized that hides latent limitations and inflexibilities. See how much you can address from there. (*4b*)
3. The camera matrix implementation looks unfinished. Right now the camera is pinned to the origin and the up vector is hardcoded. Looks like you're aware of this, but noting that I noticed it too. (*2a*, *4a*)
4. *** Looks like the general transform propagation structure is flawed as well. I see the functions for translating/scaling/rotating shapes, but I tried setting them in places to no effect. Looking at the way this is being done, with properties being propagated from the child up to the parent, I have my doubts that this is sufficiently general. As pointed out in the HW 0329a feedback, the act of flattening the array of objects loses the whole parent-child relationship *at drawing time*, and this makes the correct accumulation of transformations harder to track. Here yet again is a disadvantage of going halfway with the tree structure. (*2a*, *4a*)

2a — / …Something is off with the whole object tree/transform structure.
2b — | …Projection is successful, so expand those horizons!
3a — | …Design refinements needed on the libraries.
3d — +

CMSI 371-01
Computer Graphics
Spring 2016

Assignment 0329b Feedback

All caps are released with the outcomes in this assignment because a sufficient amount of functionality will have been reached here.

4a — **|** …Seems to be OK but some things don't work as expected. I am either seriously misunderstanding the structure of the code (which then means it requires some design thought, see below), or there are bugs in the way transforms are handled/propagated.

4b — **/** …Details already noted above.

4c — **+**

4d — **+**

4e — **+**

4f — **+** …Same notes for *4e* and *4f* as in HW 0329a.